Algorithm analysis

Final Report

A comparison of AdaBoost algorithms for time series forecast combination

Devon K. Barrowa, Sven F. Crone b

Name: Jonathan D. Freire V.

26/11/2018

Content

1. Introduction

Knowing how a certain product will be sold tomorrow or within a month helps companies manage their money and take strategies that allow them to make a profit. In many other cases, such the prediction of climate or the price of oil, it is also important to know what will happen in the future in order to take action and carry everything in a better way. In recent years it has been possible to predict all these variables thanks to time series forecasting, a method in which a set of data is analyzed in a given time and it is predicted what the new values will be for the future.

The use of machine learning to perform this process is something relatively new, but it has been very useful giving very accurate values to the real data, gaining more and more trust between companies and programmers that are dedicated to this.

That is why in this report an analysis will be made about the use of some variants of the Adaboosting algorithm for the prediction of time series, using different types of data in order to evaluate which of these algorithms is better when predict a set of data.

2. Description of the Algorithm

2.1 Explanation

The algorithm takes as input two parameters contained in the time series, then makes an initial prediction with a weak prediction model as a decision tree. This will give us a first classification of the data.

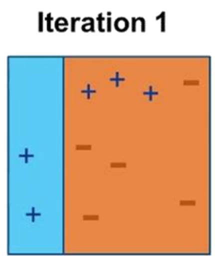

Figure 1. First Iteration of Adaboost.

Subsequently a greater weight is assigned to the data that had a bad prediction so that it can be well classified, while those that were misclassified will be given a lower weight.

Figure 2. Posterior iterations of the algorithm

Finally, all predictions are combined to obtain a much more precise model where the weights of each data have been taken into account.

Figure 3. Union of all weak predictions

2.2 Input

There are 4 input data:

X_train: matrix of 2 columns, where the first represents the month in which the sample was taken and the second represents the value that the company won in that month.

y_train: Represents the value that the company earns in a month (the same value as the second column of X_train).

X_test: Data to test the algorithm (Same scheme as X_train).

y_test: Data to test the algorithm (Same scheme as y_train)

2.3 Output

Y_pred: Value that is obtained when predicting the values of X_train

2.4 Algorithm (Original Adaboost)

1) A for cycle starts from 1 to K (where K is the number of prediction models).

2) Input data (X_i, y_i) is entered.

3) A weight vector $w_i = 1$ is created.

4) A probability of be in training data set vector is created where:

$$p\_i\text{^}k = \frac{w_i^k}{\sum w_i^k}$$

5) A prediction model $f_k = x \rightarrow y$ is built, this will return the loss suffered L_i^k where $L_i^k \in \{0,1\}$ when predicting each observation through the calculus of the error between the observed value and the predicted value.

$$err = \left|\frac{f_k(x_i) - y_i}{y_i}\right| \text{ (AdaboostRT and original)}$$

$$err = \left|\frac{f_k(x_i) - y_i}{D}\right| \text{ (AdaboostR2)}$$

(If the error is bigger than a threshold, $L_i^k = 1$, else $L_i^k = 0$)

6) Calculate the average loss

$$L_i^- = \sum L_i^k p_i^k$$

7) The accuracy of the prediction is measured with a variable β_k , where β could be:

$$\beta_k = log\frac{1-L_k}{\bar{L}_k} \quad \text{(AbaboostR2 and Original)}$$

$$\beta_k = log\frac{1}{\bar{L}_k} \quad \text{(Adaboost.RT)}$$

8) The weights of each observation *i* are updated for the next iteration *k + 1*

$$w_i^{k+1} = w_i^k \beta_k^{(1-L_i^k)}$$

9) Finally the predictions $f_k\ (x_i\ )$ of the K models are combined (added) using their weights.

2.5 Algorithm Code

```
        #Import libraries
import numpy as np
import pandas as pd

#Obtain data from the time series
dataset = pd.read_csv("C:/Users/Dell/Desktop/dx.csv")
X = dataset.iloc[:,[0,1]].values
y = dataset.iloc[:,1].values

#Save data in arrays
X_test = np.zeros((18,2))
X_train = np.zeros((len(X)-18, 2))
y_test = np.zeros(18)
y_train = np.zeros(len(X)-18)
for i in range(len(X_train)):
    X_train[i][0] = X[i][0]
    X_train[i][1] = X[i][1]
    y_train[i] = y[i]
for i in range(len(X_test)):
    X_test[i][0] = X[i+len(X_train)][0]
    X_test[i][1] = X[i+len(X_train)][1]
    y_test[i] = y[i+len(X_train)]

#Normalize the data to obtain better results
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

#Train the algorithm
from sklearn import tree
classifier = tree.DecisionTreeClassifier(criterion = 'entropy')
classifier.fit(X_train, y_train)
y_pred_inic = classifier.predict(X_test)
X_train1 = np.zeros((len(X_test),2))

for i in range(len(X_test)):
    X_train1[i][1] = X_test[i][1]
w = np.ones(len(X_test))
p = np.zeros(len(w))
LAvg = 0.6

#Start the k cycles
while LAvg >= 0.5:
```

```python

#Update the vector of weights
    for i in range(len(w)):
        p[i] = (w[i]/np.sum(w))

#Apply the weak prediction algorithm
    classifier = tree.DecisionTreeClassifier(criterion = 'entropy')
    classifier.fit(X_train1, y_pred_inic, sample_weight=p)
    y_pred = classifier.predict(X_test)


    error_vec = np.zeros(len(y_test))
    L = np.zeros(len(y_test))
    thrhold = 0.04 # Choose a value to compare the error

#OBTAIN THE LOSS SUFFERED FOR Lk
################### ADABOOST.RT AND ORIGINAL ###################
    for i in range(len(error_vec)):
      error_vec[i]=np.absolute(((y_pred[i]-y_test[i])/y_test[i]))
      if error_vec[i] > thrhold:
          L[i] = 1
      else:
          L[i] = 0
    LAvg = 0
####################################################################

########################### ADABOOST.R2 ##########################
    """
   maxi = 0.00001
   for i in range(len(error_vec)):
      D = np.absolute((y_pred[i] - y_test[i]))
      if D > maxi:
         maxi = D

   for i in range(len(error_vec)):
      error_vec[i] = np.absolute((y_pred[i] - y_test[i]))/maxi

      if error_vec[i] > thrhold:
         L[i] = 1
      else:
         L[i] = 0
   """
###################################################################


#Obtain the Average of the values of the vector Lk
    for i in range(len(L)):
        LAvg = LAvg + (L[i]*p[i])

#Obtain the predictive accuracy to calculate the new weights
    ############## ADABOOST.R2 AND ORIGINAL ADABOOST #############
    b = np.log((1 - LAvg)/LAvg)
####################################################################

    ######################## ADABOOST.RT ########################
   #b = np.log(1/LAvg)
####################################################################

#Calculate the weights fot the next iteration
    for i in range(len(w)):
```

```
        w[i] = (w[i] * pow(b, 1-L[i]))

#Print the results
print("Y de testeo")
for i in range(len(y_test)):
    print(y_test[i])
print("Y de prediccion")
for i in range(len(y_pred)):
    print(y_pred[i])
```

*The algorithm was programmed in Python using “Spyder” in order to can use libraries like “numpy” and “pandas” to make easiest and fast the compiling and programming process.

2.5.1 System characteristics:

Memory (RAM): 16 GB

Processor: Intel Core i7

Operative System: Windows 10 Home

3. Results

3.1 Input Values

nnnn Observation of Training Data

??? Test Data to be forecasted - now filled with true values

Month	Company 1	Company 2	Company 3	Company 4	Company 5
1	5520	4670	13430	1860	3270
2	3940	3150	13020	3880	3690
3	4490	3770	11710	2660	4360
4	5030	3380	9265	3080	3520
5	5660	3930	7280	3300	4140
6	4790	3350	5040	6300	3440
7	5520	3610	3860	3140	3750
8	5560	3740	6160	2940	4140
9	5200	3590	13610	4940	5610
10	6670	3730	15455	4240	5350
11	5900	3850	14530	3360	4270
12	5280	4010	13815	5400	3160
13	6490	4790	12860	3280	2560
14	5560	3330	11500	3780	3410
15	5090	2940	10660	4400	3670
16	7090	4020	9340	4520	3670
17	6570	4010	8050	4820	3200
18	5890	3450	6540	3040	3700
19	6640	4080	5060	3940	3430
20	6360	3590	6350	2820	4130
21	5640	3500	14130	2860	4870
22	6630	3680	16380	3860	6310
23	5330	3750	16160	2900	4680
24	5540	4370	15850	4960	2380
25	7100	4180	15930	5300	3430
26	5410	3350	15320	5060	2950
27	5800	3780	13420	5300	3130
28	6110	3490	12255	4700	3650
29	6870	3520	8785	5300	2980
30	6560	3880	6380	4900	3720
31	6120	4430	4760	5840	2960
32	6540	3420	5730	3800	4300
33	5810	3750	10810	5240	5550
34	6300	4010	12845	3720	4380
35	6270	3940	12865	4300	3620
36	6900	4330	13515	6620	3350
37	6310	4160	13880	4840	3300
38	5330	3520	12960	5560	3800
39	5700	3710	12090	7340	3420
40	6680	3650	9510	6240	3860
41	5510	3460	8130	7200	2900
42	6690	3770	6625	4500	3240
43	5870	4140	4920	3360	3150
44	7140	3470	4650	4100	4460
45	6680	3730	10085	4260	5040
46	6520	3590	13960	4820	4660
47	6020	3970	14495	5280	3450
48	6190	4220	14340	4160	2660
49	6690	5050	13875	4340	2400
50	6330	4090	13135	4680	2200
51	7620	3630	13415	3620	3020
52	5430	3470	9280	4420	3450
53	5410	3350	7075	6000	4110
54	6030	3740	5660	4880	2760
55	5740	3370	4270	3360	3390
56	6520	3320	5085	7340	3740
57	6080	3750	11945	2900	4830
58	5990	3210	14335	3920	4450
59	6750	3870	14105	1800	3140
60	6770	4270	13755	2780	4700
61	6320	4200	12920	2180	2330
62	5960	3790	11650	2860	2740
63	6190	3920	10720	4440	2480
64	5250	3480	8600	4040	2690
65	5910	3160	7795	3980	5850
66	6430	4070	6550	4980	2450
67	5950	3420	4800	3260	3720
68	5060	3810	5900	4420	2780
69	5400	2950	14095	5660	5690
70			15170	4180	4150
71			14875	3420	4460
72			15230	3260	2730
73			13685	2720	3580
74			12780	2000	2870
75			11510	4200	4210
76			9915	3920	4500
77			8740	3820	3560
78			7870	3920	3800
79			6650	2840	3260
80			5285	3480	3410
81			13195	5140	3860
82			13390	3000	5000
83			13490	3120	3840
84			13445	2780	3210
85			13070	2140	2930
86			12480	2640	3630
87			11550	3720	2760
88			10725	2860	3310
89			9130	3220	4320
90			7885	2680	3010
91			6415	3040	3440
92			5540	3440	5100
93			9350	4440	4170
94			12645	3780	4540
95			11985	2640	3410
96			10055	2560	2590
97			10295	2520	2640
98			10280	1480	2970
99			9420	2260	3000
100			9575	1880	2900
101			8090	1920	3010
102			5855	4080	3380
103			4445	5480	3060
104			3555	7060	3100
105			12870	7580	3940
106			14750	5120	5360
107			13615	4040	3280
108			13705	2940	3190
109			13940	3120	4950
110			11900	3720	2910
111			9000	3320	2820
112			7340	3620	3050
113			6425	5080	2930
114			5535	2300	2800
115			4050	5280	3220
116			3485	4300	3120
117			8090	3980	4000
118			11380	5040	4840
119			11355	4700	2630
120			10530	3160	2920
121			9285	3720	2350
122			9350	2720	2850
123			8300	3720	2700
124			8090	3020	2870
125			6670	4400	2580
126			5205	2520	2450
127			3645	3380	2950
128			3595	2280	3350
129			10135	3260	4450
130			11385	4700	4210
131			10445	4760	2650
132			9520	5480	2414
133			8940	4620	2823
134			5885	4240	2444
135			7690	6160	2666
136			6245	4120	3052
137			5560	6900	2397
138			4850	4020	2463
139			3830	5400	3385
140				3560	3290
141				5460	5408
142				5580	4000
143				5220	2513
144				5200	

Figure 4. Input values.

3.2 Output Values:

Original Adaboost Algorithm

	Company 1 51 training data			Company 2 51 training data			Company 3 121 training data			Company 4 126 training data			Company 5 126 training data SPECIAL CASE		
Month	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error
1	543	541	2	3470	3470	0	9350	9285	65	3380	3320	60	2450	2580	130
2	541	541	0	3350	3350	0	8300	8090	210	2280	2300	20	2950	2950	0
3	603	602	1	3740	3730	10	8090	8090	0	3260	3260	0	3350	3380	30
4	574	581	7	3370	3380	10	6670	6425	245	4700	4700	0	4450	4450	0
5	652	652	0	3320	3330	10	5205	5535	330	4760	4820	60	4210	4170	40
6	608	611	3	3750	3750	0	3645	3485	160	5480	5480	0	2650	2630	20
7	599	602	3	3210	3330	120	3595	3485	110	4620	4680	60	2414	2580	166
8	675	669	6	3870	3880	10	1014	1028	14	4240	4260	20	2823	2850	27
9	677	669	8	4270	4220	50	1139	1138	1	6160	7060	900	2444	2580	136
10	632	633	1	4200	4220	20	1045	1053	8	4120	4080	40	2666	2700	34
11	596	602	6	3790	3880	90	9520	9575	55	6900	7060	160	3052	3050	2
12	619	619	0	3920	3930	10	8940	9285	345	4020	4040	20	2397	2580	183
13	525	533	8	3480	3470	10	5885	5855	30	5400	5480	80	2463	2580	117
14	591	587	4	3160	3330	170	7690	7340	350	3560	3620	60	3385	3380	5
15	643	652	9	4070	4090	20	6245	5855	390	5460	5480	20	3290	3280	10
16	595	602	7	3420	3420	0	5560	5535	25	5580	5560	20	5408	5360	48
17	506	533	27	3810	3880	70	4850	5535	685	5220	5240	20	4000	4000	0
18	54	57	3	2950	2940	10	3830	3485	345	5200	5240	40	2513	2580	67
Average			5,27777778			33,8888889			187,111111			87,7777778			56,3888889

Figure 5. Output values, Original Adaboost.

Adaboost.RT Algorithm

	Company 1 51 training data			Company 2 51 training data			Company 3 121 training data			Company 4 126 training data			Company 5 126 training data SPECIAL CASE		
Month	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error
1	543	541	2	3470	3470	0	9350	9285	65	3380	3360	20	2450	2580	130
2	541	541	0	3350	3350	0	8300	8600	300	2280	2300	20	2950	2930	20
3	603	602	1	3740	3730	10	8090	8090	0	3260	3260	0	3350	3380	30
4	574	581	7	3370	3380	10	6670	6425	245	4700	4700	0	4450	4450	0
5	652	652	0	3320	3330	10	5205	5285	80	4760	4820	60	4210	4320	110
6	608	602	6	3750	3750	0	3645	3485	160	5480	5480	0	2650	2700	50
7	599	602	3	3210	3150	60	3595	3485	110	4620	4680	60	2414	2580	166
8	675	669	6	3870	3880	10	1014	1028	14	4240	4200	40	2823	2850	27
9	677	669	8	4270	4220	50	1139	1136	3	6160	6240	80	2444	2580	136
10	632	633	1	4200	4220	20	1045	1053	8	4120	4080	40	2666	2700	34
11	596	602	6	3790	3780	10	9520	9575	55	6900	7060	160	3052	3050	2
12	619	633	14	3920	3940	20	8940	9000	60	4020	4040	20	2397	2580	183
13	525	533	8	3480	3470	10	5885	5900	15	5400	5480	80	2463	2580	117
14	591	602	11	3160	3150	10	7690	7340	350	3560	3620	60	3385	3380	5
15	643	652	9	4070	4090	20	6245	5900	345	5460	5480	20	3290	3280	10
16	595	602	7	3420	3420	0	5560	5540	20	5580	5480	100	5408	5360	48
17	506	533	27	3810	3780	30	4850	5285	435	5220	5240	20	4000	4000	0
18	54	57	3	2950	2940	10	3830	4050	220	5200	5240	40	2513	2580	67
Prom			6,61111111			15,5555556			138,055556			45,5555556			63,0555556

Figure 6. Output values, Adaboost.RT.

Adaboost.R2 Algorithm

	Company 1 51 training data			Company 2 51 training data			Company 3 121 training data			Company 4 126 training data			Company 5 126 training data SPECIAL CASE		
Month	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error	Test Value	Pred, Value	Error
1	543	533	10	3470	3470	0	9350	9285	65	3380	3320	60	2450	2450	0
2	541	533	8	3350	3350	0	8300	8740	440	2280	2300	20	2950	2920	30
3	603	611	8	3740	3730	10	8090	8090	0	3260	3260	0	3350	3350	0
4	574	581	7	3370	3380	10	6670	6650	20	4700	4700	0	4450	4450	0
5	652	611	41	3320	3330	10	5205	5535	330	4760	4820	60	4210	4320	110
6	608	611	3	3750	3750	0	3645	3485	160	5480	5480	0	2650	2630	20
7	599	602	3	3210	3330	120	3595	3485	110	4620	4680	60	2414	2450	36
8	675	669	6	3870	3880	10	1014	1028	14	4240	4200	40	2823	2850	27
9	677	669	8	4270	4220	50	1139	1138	1	6160	5480	680	2444	2450	6
10	632	611	21	4200	4160	40	1045	1053	8	4120	4100	20	2666	2700	34
11	596	602	6	3790	3780	10	9520	9575	55	6900	7060	160	3052	3050	2
12	619	611	8	3920	3930	10	8940	9000	60	4020	4040	20	2397	2350	47
13	525	533	8	3480	3470	10	5885	5855	30	5400	5480	80	2463	2450	13
14	591	581	10	3160	3330	170	7690	7340	350	3560	3620	60	3385	3380	5
15	643	611	32	4070	4090	20	6245	5855	390	5460	5480	20	3290	3220	70
16	595	589	6	3420	3420	0	5560	5535	25	5580	5480	100	5408	5360	48
17	506	509	3	3810	3780	30	4850	5535	685	5220	5280	60	4000	4000	0
18	54	57	3	2950	2940	10	3830	4050	220	5200	5120	80	2513	2450	63
Prom			10,6111111			28,3333333			164,611111			84,4444444			28,3888889

Figure 7. Output values, Adaboost-R2.

3.3 Comparison

After performing the corresponding tests with the 5 different types of data in the 3 different types of algorithms, it was possible to determine that the Adaboosting.RT variant is the most efficient, since in most tests it had greater accuracy when predicting the data.

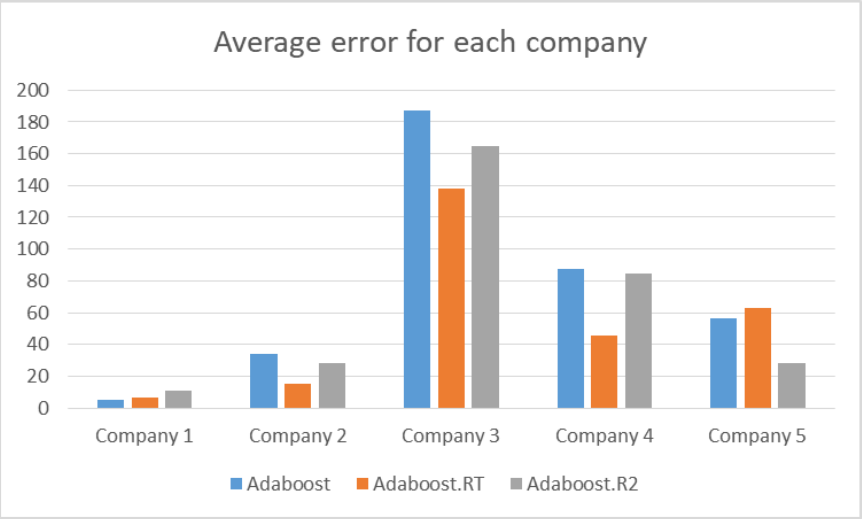

Figure 8. Comparison of errors.

Also It is possible to see that the original Adaboost algorithm have the biggest error of those three, so it can be concluded that when you change the parameters we could have better results for an specific case.

Conclusions

- In conclusion it was possible to see that the Adaboost.RT algorithm can give better forecasting among the 3 variants of Adaboosting, this can be observed also in the paper. This is because the predictive accuracy formula works better for this type of problems.

- Another observation is that the prediction model can give unexpected values, causing the algorithm to take longer without having to do with the type of algorithm that is being used, which is why the execution time was not taken into account as a measurement of the efficiency of the algorithms.

References

- Shrestha, D. L., & Solomatine, D. P. (2006). Experiments with AdaBoost.RT, an improved boosting scheme for regression. Neural Computation, 18(7), 1678–1710.
- Devon K. Barrow, Sven F. Crone. (2006) A comparison of AdaBoost algorithms for time series forecast combination, International Journal of Forecasting, Volume 32, Issue 4, Pages 1103-1119, ISSN 0169-2070